SŠPU
STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE
dokumentace

Snap2Doc
Flutter aplikace pro digitalizaci knih

Marek Samel

Obor:	18-20-M/01 INFORMAČNÍ TECHNOLOGIE se zaměřením na počítačové sítě a programování
Třída:	IT4
Školní rok:	2024/2025

Poděkování

Rád bych poděkoval Tomáši Schablickému za cennou pomoc a své přítelkyni Vanesse za její nekonečnou podporu.

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 31. 12. 2024

podpis autora práce

ABSTRAKT

Aplikace Snap2Doc je navržena pro převod obrázků s textovým obsahem do dokumentové podoby. Motivací pro její vytvoření byla absence snadného nástroje pro digitalizaci tištěných knih či dokumentů pro čtečky e-knih. Uživatel může importovat fotografie z galerie nebo je pořizovat přímo vestavěným fotoaparátem. Pomocí OCR technologie Google ML Kit aplikace extrahuje text a převádí jej do dokumentového formátu.

Pro optimalizaci výkonu aplikace generuje náhledy obrázků s nižším rozlišením a data ukládá do cache, která se při každém spuštění automaticky vymazává. Snap2Doc byla vyvinuta ve frameworku Flutter, což je moderní framework pro vývoj multiplatformních aplikací. Uživatelské rozhraní je navrženo pro jednoduchost a obsahuje vizuální zpětnou vazbu v podobě animací načítání.

ABSTRACT

The Snap2Doc application is designed to convert images with text content into document form. The motivation for its creation was the lack of an easy tool for digitizing printed books or documents for e-book readers. Users can import photos from their gallery or take them directly with the built-in camera. Using Google ML Kit OCR technology, the application extracts text and converts it into a document format.

To optimize performance, the application generates lower-resolution image previews and stores data in a cache that is automatically cleared each time the app is launched. Snap2Doc was developed using the Flutter framework, a modern framework for creating cross-platform applications. The user interface is designed for simplicity and includes visual feedback in the form of loading animations.

Obsah

ÚVOD

Tato práce se zabývá vývojem mobilní aplikace Snap2Doc, která poskytuje efektivní způsob převodu obrázků s textovým obsahem na digitální dokumenty, konkrétně ve formátu PDF. Cílem projektu je nabídnout uživatelům nástroj pro snadnou správu a digitalizaci tištěných dokumentů, knih nebo fotografií, a to prostřednictvím intuitivního a uživatelsky přívětivého rozhraní.

Problém, kterému se aplikace věnuje, spočívá v nedostatečné dostupnosti specializovaných nástrojů, které by umožnily snadné zpracování tištěného obsahu a jeho převod do elektronické podoby. V důsledku toho uživatelé často hledají alternativní řešení, která nejsou dostatečně efektivní. Snap2Doc přináší jednoduchý proces, který kombinuje možnost importu fotografií z galerie nebo pořízení nových prostřednictvím vestavěného fotoaparátu, s následným zpracováním pomocí OCR technologie od Google ML Kitu pro převod textu.

Hlavním cílem této práce je detailně popsat návrh a implementaci této aplikace, zdůraznit použité technologie a jejich vzájemnou integraci, a rovněž se zaměřit na uživatelskou zkušenost (UX) a optimalizaci výkonu.

Tento úvod slouží jako základní vhled do problematiky digitalizace textu pomocí mobilních aplikací. Následující kapitoly podrobněji popisují návrh aplikace, technologické aspekty, implementaci a testování, čímž poskytují komplexní pohled na celý proces vývoje Snap2Doc.

1 VYUŽITÉ TECHNOLOGIE

Aplikace je napsaná v jazyce Dart ve frameworku Flutter. Dart jsem zvolil hlavně kvůli jeho modernosti a jednoduchosti užití.

1.1 Flutter Framework

Flutter je open-source framework vyvinutý společností Google, který umožňuje tvorbu multiplatformních aplikací pomocí jednotného kódu.

Prvním krokem pro založení nového projektu ve Flutteru je instalace Flutter SDK na místní počítač. Postupujte podle oficiální dokumentace Flutteru pro instalaci a nastavení vývojového prostředí. Po úspěšné instalaci se nastaví cesta k Flutter SDK a ověří se správnost instalace pomocí příkazu `flutter doctor`.

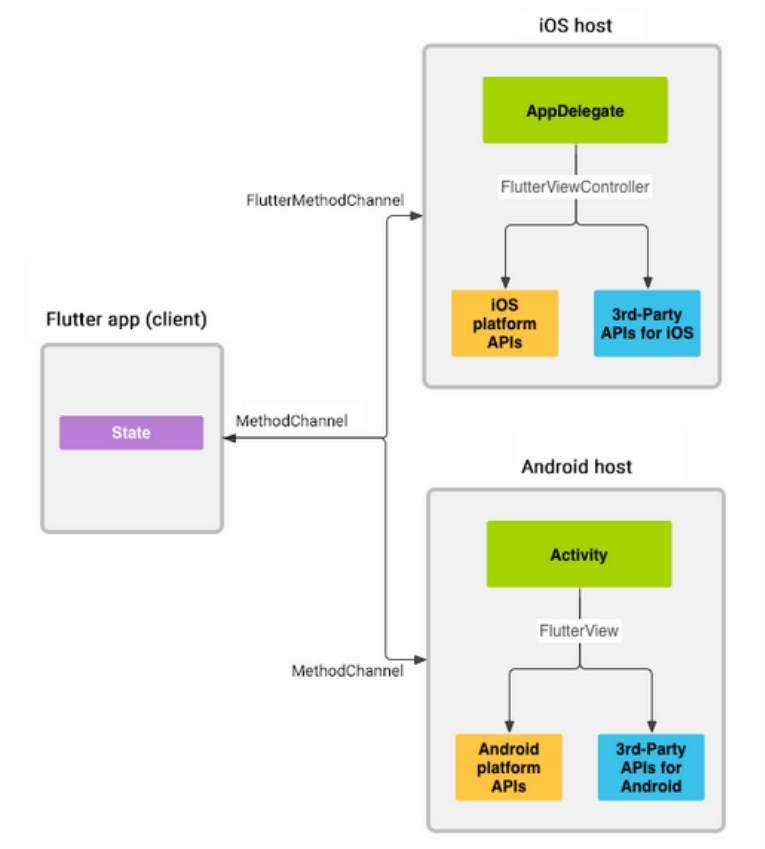
Pro vytvoření projektu ve Visual Studio Code lze využít zkratku `ctrl + shift + p` a zvolit možnost vytvoření nového projektu, což vytvoří základní strukturu pro vývoj aplikace, včetně hlavního souboru main.dart, což je vstupní bod aplikace a konfiguračního souboru pubspec.yaml, který slouží pro import knihoven. Následně můžete do složky /lib/ přikládat další soubory s kódem.

```
flutter_project/
│
├── android/          # Konfigurace a zdrojové soubory pro Android aplikaci (nativní část projektu).
├── ios/              # Konfigurace a zdrojové soubory pro iOS aplikaci (nativní část projektu).
├── lib/              # Hlavní adresář, kde se nachází zdrojový kód aplikace napsaný v Dart.
│   ├── main.dart     # Vstupní bod aplikace (hlavní soubor aplikace).
│
├── test/             # Testovací soubory aplikace napsané v Dart.
│   └── widget_test.dart  # Ukázkový test widgetu.
│
├── build/            # Automaticky generované soubory během sestavení aplikace (needitujte ručně).
├── web/              # Podpora pro webovou verzi aplikace (volitelná).
├── macos/            # Podpora pro macOS aplikaci (volitelná).
├── windows/          # Podpora pro Windows aplikaci (volitelná).
├── linux/            # Podpora pro Linux aplikaci (volitelná).
│
├── .dart_tool/       # Interní data nástrojů Dart (automaticky generované, needitujte).
├── .idea/            # Nastavení projektu pro IntelliJ IDEA (volitelné).
├── .vscode/          # Nastavení projektu pro Visual Studio Code (volitelné).
│
├── pubspec.yaml      # Konfigurační soubor projektu, obsahuje informace o balíčcích, verzích a assetech.
├── pubspec.lock      # Zámkový soubor pro balíčky (automaticky generovaný).
├── README.md         # Popis projektu, často obsahuje základní informace a instrukce.
└── analysis_options.yaml  # Nastavení statické analýzy pro Dart (volitelné).
```

1.1) Adresářová struktura Flutter projektu

1.2 OCR

Pro implementaci funkce rozpoznávání textu (OCR) v aplikaci byla využita technologie Google ML Kit. Tento nástroj poskytuje širokou škálu funkcí strojového učení, které umožňují efektivní a přesné zpracování obrázků s textovým obsahem. Google ML Kit je ideální volbou díky vysoké přesnosti a volitelné podpoře různých jazyků, což je klíčové pro zajištění univerzálního rozpoznávání textu. Klíčovými vlastnostmi této technologie jsou její schopnost pracovat offline a zpracovávat obrázky různých formátů (JPEG, PNG, BMP). Google ML Kit zajišťuje vysokou úroveň přesnosti, což je zásadní pro zpracování textu v různých stylech písma, speciálních znacích nebo textu v neobvyklých abecedách. Díky těmto schopnostem je aplikace schopna poskytovat uživatelům spolehlivý nástroj pro digitalizaci tištěného obsahu.



1.2) Diagram fungování Google ML Kit Text Recognition

2 ZPŮSOBY ŘEŠENÍ A POUŽITÉ POSTUPY

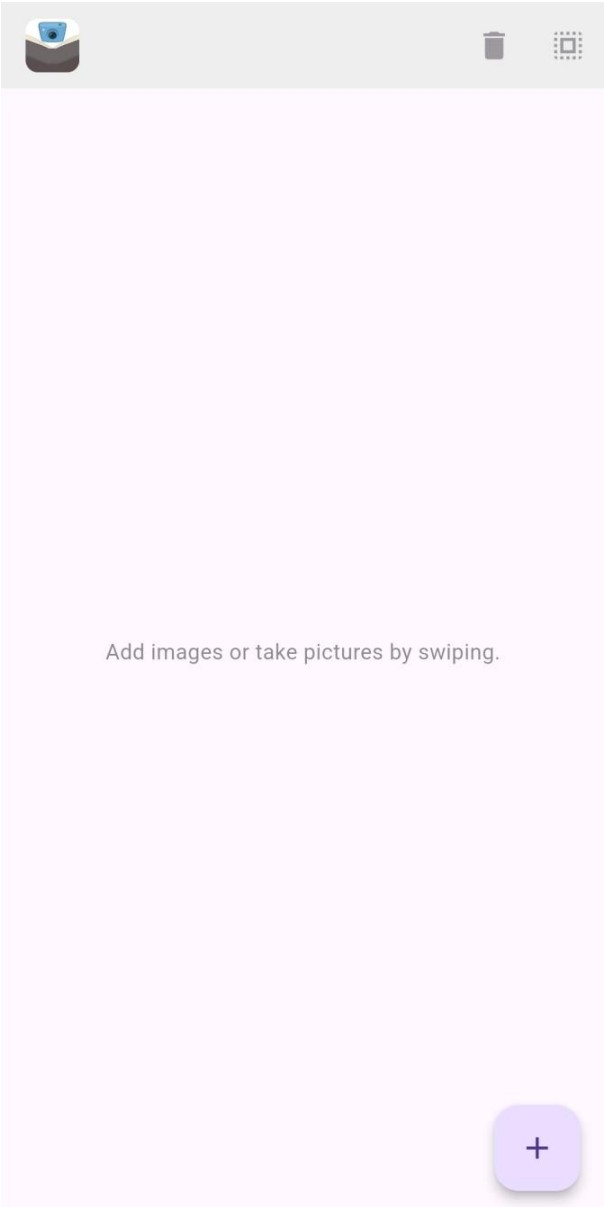
V této kapitole jsou popsány funkce a řešení problematiky vývoje aplikace.

2.1 Stránka galerie

Stránka galerie zároveň slouží jako hlavní menu aplikace, v němž se uživatel může snadno a intuitivně pohybovat. Nabízí rychlý přístup ke všem klíčovým funkcím aplikace, čímž zajišťuje plynulý a efektivní pracovní postup. Mezi dostupné funkce patří:

- **Zobrazení nahraných obrázků**: Uživatel má okamžitý přístup ke svým souborům přímo v galerii.
- **Označování fotografií**: Umožňuje vybrat konkrétní snímky pro další zpracování.
- **Tlačítko pro přidání fotografií přes prohlížeč**: Uživatel může nahrávat nové fotografie přímo z úložiště zařízení.
- **Tlačítko pro uložení dokumentu**: Po označení fotografií lze snadno spustit proces převodu na dokument.
- **Fotoaparát přístupný gestem potáhnutí doprava**: Minimalizuje vizuální zátěž rozhraní a zajišťuje rychlý přístup k fotoaparátu.
- **Označení všech fotografií jedním kliknutím**: Zjednodušuje hromadnou práci s obrázky.
- **Možnost smazání označených fotografií**: Poskytuje jednoduchý způsob správy obsahu.

Stránka galerie je navíc obohacena o animace, které zpříjemňují uživatelskou zkušenost a zdůrazňují interaktivní prvky. Tyto vizuální efekty zároveň zvyšují přehlednost a pomáhají uživateli lépe se orientovat v aplikaci.

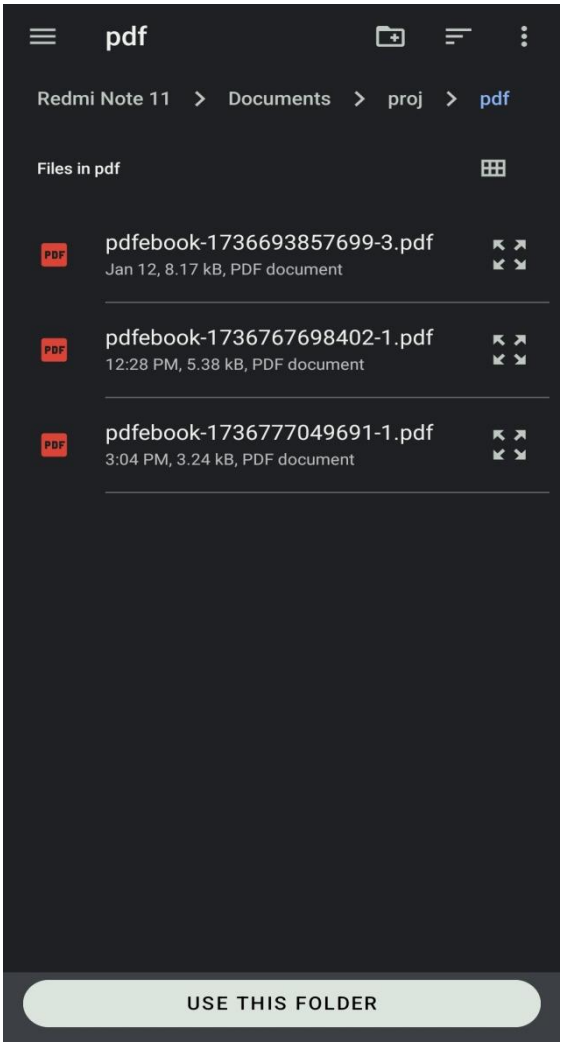


2.1) Prázdná galerie

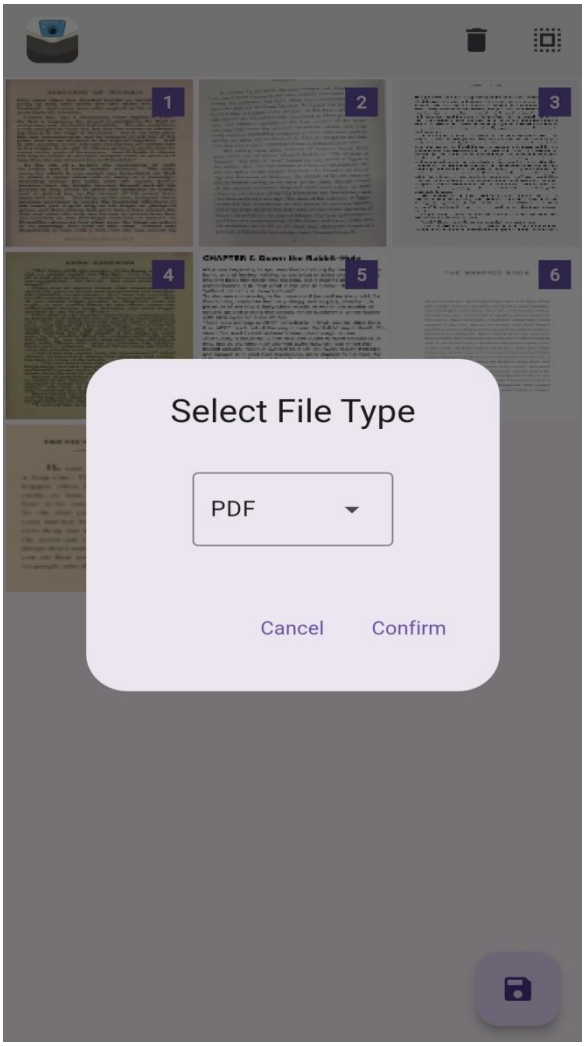
2.2) Naplněná galerie

2.1.1 Ukládání vygenerovaného dokumentu

Po označení fotografií, které si uživatel přeje převést na dokument, se funkce tlačítka „Přidat z prohlížeče“ dynamicky změní na tlačítko „Uložit jako dokument“. Po jeho stisknutí je uživatel přesměrován do průzkumníku souborů, kde má možnost zvolit cílovou složku pro uložení nově vytvořeného dokumentu. Následně je aplikací vyzván k výběru požadovaného formátu dokumentu, čímž je zajištěna flexibilita a kompatibilita výstupu s dalšími nástroji či požadavky. Tento postup je navržen tak, aby byl intuitivní a plynulý, přičemž minimalizuje počet nutných kroků při práci s aplikací.



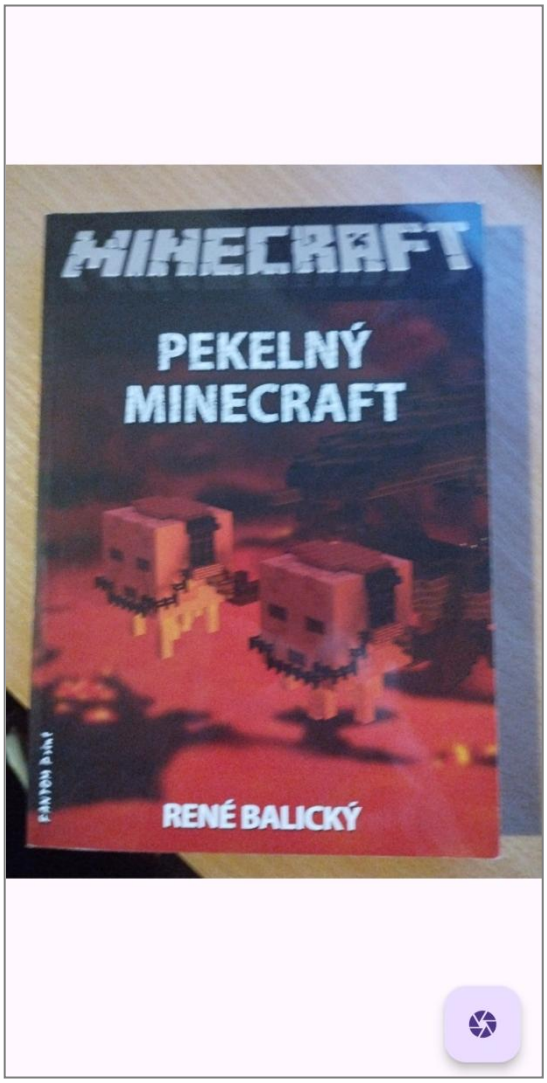
2.3) Průzkumník souborů



2.4) Výběr formátu

2.2 Stránka fotoaparátu

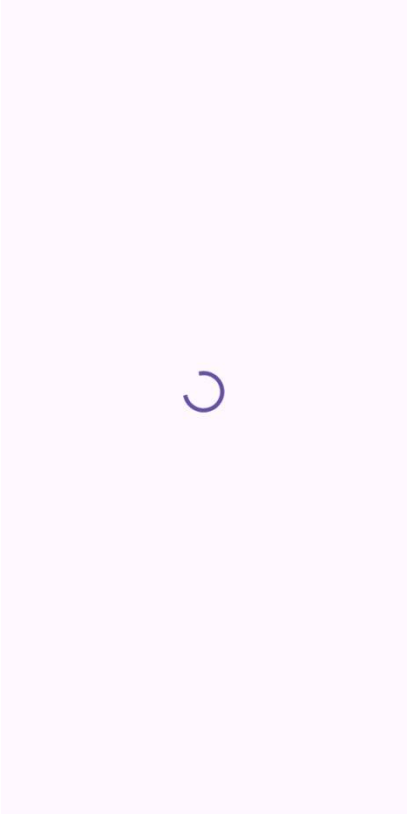
Integrovaný fotoaparát aplikace je inicializován již při jejím spuštění, což eliminuje jakékoli zpoždění při jeho následném používání. Tento přístup zajišťuje, že uživatel nemusí čekat na jeho načtení, což přispívá k rychlejší a efektivnější práci s aplikací. Při otevření je fotoaparát automaticky nastaven na zadní kameru, která je ve většině případů primárně využívána pro snímání dokumentů či dalších objektů, což zjednodušuje použití a odpovídá očekávanému chování aplikace.



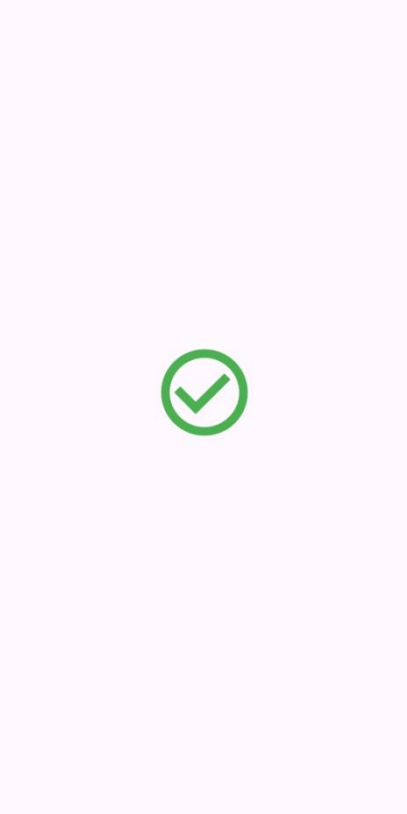
2.5) Integrovaný fotoaparát

2.3 Stránka procesování dokumentu

Po úspěšném výběru požadované složky a specifikace typu dokumentu je uživatel automaticky přesměrován na stránku určenou k procesování dokumentu. Na této stránce uživatel vyčkává na dokončení procesu zpracování. Po jeho ukončení, v závislosti na úspěšnosti provedení, se zobrazí příslušná ikona informující o výsledku operace, a to po dobu dvou sekund. Následně je uživatel automaticky přesměrován zpět do galerie, kde může pokračovat v práci s aplikací. Tímto způsobem je zajištěna intuitivní a plynulá uživatelská zkušenost.



2.6) Probíhající zpracování



2.7) Úspěšné zpracování

2.8) Neúspěšné zpracování

2.4 Příklady kódu a problematika

V následujících podkapitloách jsou popsány některé potíže na které jsem narazil a následně i vyřešil, s praktickými ukázkami kódu.

2.4.1 Řešení přístupu stránky foťáku

Tento problém vznikl jako důsledek specifických požadavků na grafický design aplikace, který kladl důraz na minimalistické a přehledné uživatelské rozhraní. Konkrétně šlo o otázku, jak umožnit uživatelům rychlý a intuitivní přístup k funkci fotoaparátu, aniž by bylo nutné aplikaci zahlcovat nadměrným množstvím tlačítek či jiných vizuálních prvků. Tento požadavek byl elegantně vyřešen implementací gesta potáhnutí prstem směrem doprava, které uživatele plynule přesměruje přímo na funkci fotoaparátu. Tento přístup nejen šetří místo na obrazovce, ale také přispívá k plynulému a modernímu uživatelskému zážitku.

```
      body: GestureDetector(
        behavior: HitTestBehavior.translucent,
        onPanUpdate: (details) {
          if (details.delta.dx > 0) {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) =>
CameraScreen()),
            ).then((_){
              loadImages();
            });
          }
        },
```

2.9) Widget detekující gesta

2.4.2 Optimalizace zobrazení

V původní implementaci mého kódu byly do miniatur načítány celé nahrané obrázky v jejich původní velikosti a kvalitě. Tento přístup vedl k výrazným výkonnostním problémům, zejména při práci s větším množstvím obrázků. Tento problém byl efektivně vyřešen vytvořením miniatur s nižší kvalitou a rozlišením, které výrazně snižují náročnost na systémové zdroje a zlepšují celkovou výkonost aplikace, aniž by byla obětována použitelnost miniatur.

```
Future<void> loadImages() async {
  ↑/ předchozí kód
  // část funkce obstarávající miniatury
  final List<Uint8List> tempLowResImages = [];
  for (final image in tempImages) {
    final bytes = await image.readAsBytes();
    final codec =
    await instantiateImageCodec(bytes,
        targetWidth: 100, targetHeight: 100);
    final frame = await codec.getNextFrame();
    final lowResBytes =
    (await frame.image.toByteData(
    format: ImageByteFormat.png))!
            .buffer
            .asUint8List();
    tempLowResImages.add(lowResBytes);
  }
  /↓ následující kód
}
```

2.10) Výtažek z funkce

2.4.3 Třída pro OCR

Pro zajištění snadné a přehledné implementace v hlavním kódu jsem považoval za vhodné navrhnout a vytvořit samostatnou třídu. Tato třída obsahuje metody, které jsou zodpovědné za obsluhu a správu funkcionalit souvisejících s OCR službou (optické rozpoznávání znaků). Tímto způsobem je zajištěna modularita kódu, jeho lepší čitelnost a snazší údržba, přičemž všechny operace související s OCR službou jsou centralizovány v jednom logickém celku.

```
class Ocr {
  final _textDetector = TextRecognizer();
  Future<String> extractText(final String imagePath)
async {
    try {
      final inputImage = InputI-
mage.fromFilePath(imagePath);
      final recognizedText =
          await _textDetector.processImage(inputImage);
      String text = '';
      for (TextBlock block in recognizedText.blocks) {
        text += '${block.text}\n';
      }
      return (text != '') ? text : 'No text found';
    } catch (e) {
      return '';
    }
  }
  Future<void> dispose() async {
    await _textDetector.close();
  }
}
```

2.11) Třída OCR

3 ZÁVĚR

Hlavním cílem tohoto projektu bylo vytvoření řešení, které umožní převod textu z obrázkových souborů do podoby dokumentu, jenž je čitelný prostřednictvím elektronické čtečky knih. Tento cíl jsem splnil, ačkoli konečný výsledek zcela neodpovídá mým původním očekáváním a ambicím. V rámci aplikace je zajištěna funkcionalita pro ukládání obrázků do dočasného úložiště, integrace fotoaparátu, a rovněž převod pořízených fotografií do formátu PDF. S těmito funkcemi jsem velmi spokojen, neboť představují zásadní část zamýšlené funkcionality.

Přesto projekt vykazuje určité nedostatky, které stojí za zmínku a které mohou mít vliv na uživatelský komfort a efektivitu celého řešení. Prvním problémem je nižší rychlost zpracování textu technologií OCR (Optical Character Recognition), což se stává zjevné zejména při práci s větším množstvím fotografií. Druhou slabinou je absence možnosti převodu obrázků přímo do formátu EPUB, jenž je preferovaným formátem pro elektronické knihy.

Navržené řešení těchto problémů spočívá v několika krocích. Rychlost OCR by mohla být zlepšena implementací multithreadingu, což by umožnilo paralelní zpracování více fotografií a výrazně zkrátilo dobu potřebnou k dokončení procesu. Pokud jde o podporu formátu EPUB, navrhuji vyvinout vlastní třídu pro ukládání dokumentů do tohoto formátu, neboť dostupné knihovny, které jsem měl možnost vyzkoušet, nejsou schopny správně zapisovat data do formátu EPUB.

I přes uvedené nedostatky považuji projekt za úspěšný v základním splnění jeho cílů a vidím v něm prostor pro další rozvoj a optimalizaci, což by mohlo vést k jeho širšímu uplatnění v praxi.

SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ

[1] *Building your first Flutter App - with a Codelab!* [online]. Dostupné na: https://www.youtube.com/watch?v=8sAyPDLorek

[2] *Flutter package Dokumentace.* [online]. Dostupné na: https://pub.dev/

[3] *Flutter Instalační Dokumentace.* [online]. Dostupné na: https://docs.flutter.dev/get-started/install

[4] *Async vs Isolates | Decoding Flutter*. [online]. Dostupné na: https://www.youtube.com/watch?v=5AxWC49ZMzs

[5] *Google's ML Kit Text Recognition for Flutter*. [online]. Dostupné na: https://pub.dev/packages/google_mlkit_text_recognition